PostgreSQL

DBA Training

Surendra Panpaliya

Agenda

DAY 1 — PostgreSQL Foundations, Architecture & Tools

DAY 2 — Transactions, Locking & Maintenance

DAY 3 — Query Optimization & Indexing

DAY 4 — Backup, Replication Concepts & Migration

A. PostgreSQL Introduction

- PostgreSQL capabilities overview
- PostgreSQL vs Oracle vs SQL Server vs AWS RDS PostgreSQL
- Architecture differences (Local vs RDS)
- MVCC high-level concept
- Key differences vs enterprise databases

A. PostgreSQL Introduction

Fidelity Use Case

PostgreSQL used for

Investment reporting, portfolio views, microservices

Oracle retained for **core settlement / legacy systems**

RDS PostgreSQL used for **managed production workloads**

B. PostgreSQL Local Architecture (1 hour)

Cluster structure

(data directory, instance, databases, schemas)

Key processes

postmaster, autovacuum, WAL writer, checkpointer

B. PostgreSQL Local Architecture (1 hour)

- Memory architecture
- WAL, checkpoints, background writer
- MVCC row versioning

Fidelity Use Case

- Gurugram supports multi
- DB clusters → shared memory understanding critical
- UK latency SLAs → WAL & checkpoint behavior matters
- China read-heavy compliance queries → MVCC benefits

C. Tools Overview & Connectivity (30 minutes)

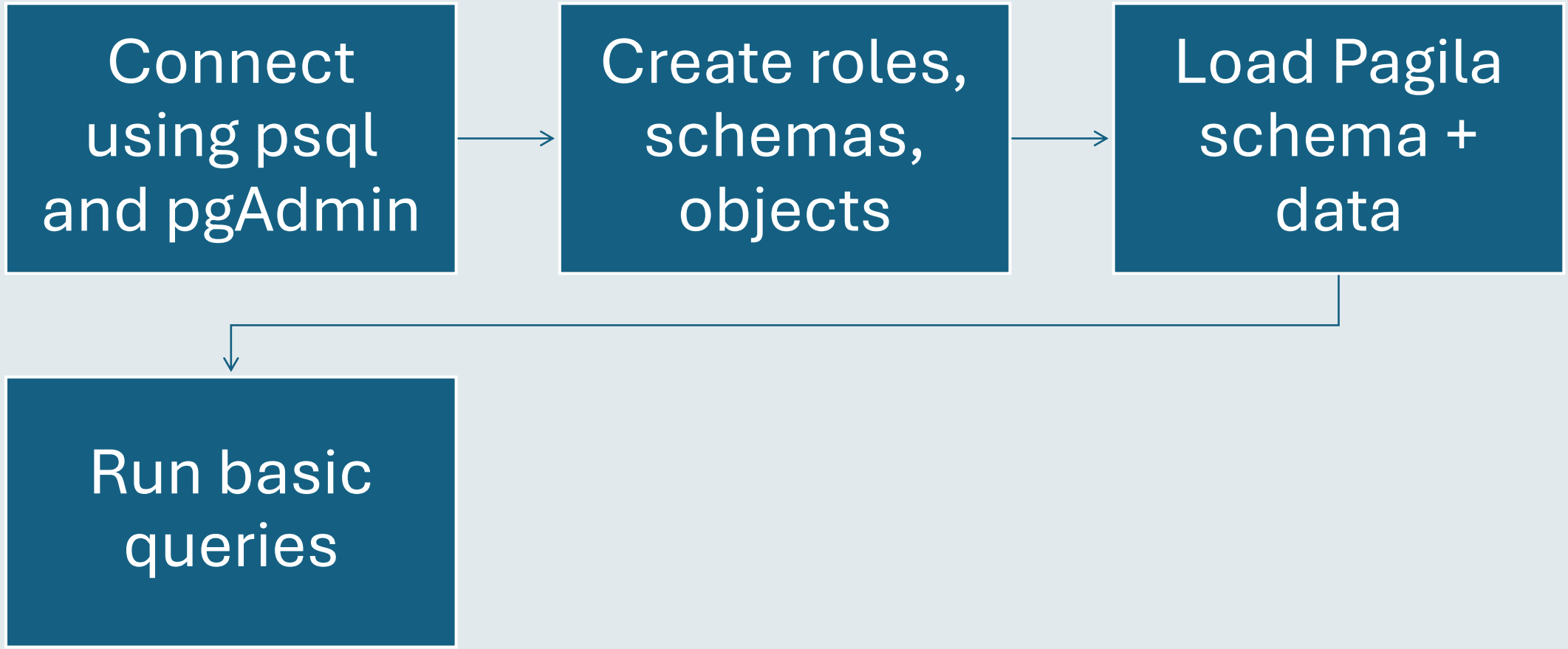
psql vs pgAdmin (DBA usage patterns)

Connecting locally

Roles, databases, schemas

Pagila dataset overview

D. Hands-On Lab



D. Hands-On Lab

Measure performance using:	\timing	EXPLAIN / EXPLAIN ANALYZE
Explore system catalogs:	pg_class	pg_stat_activity

Module-wise FAQ — Day 1

Q1. Is PostgreSQL production-ready for financial systems?

Q2. How is PostgreSQL different from Oracle RAC?

Q3. Why do DBAs still need architecture knowledge in RDS?

Agenda

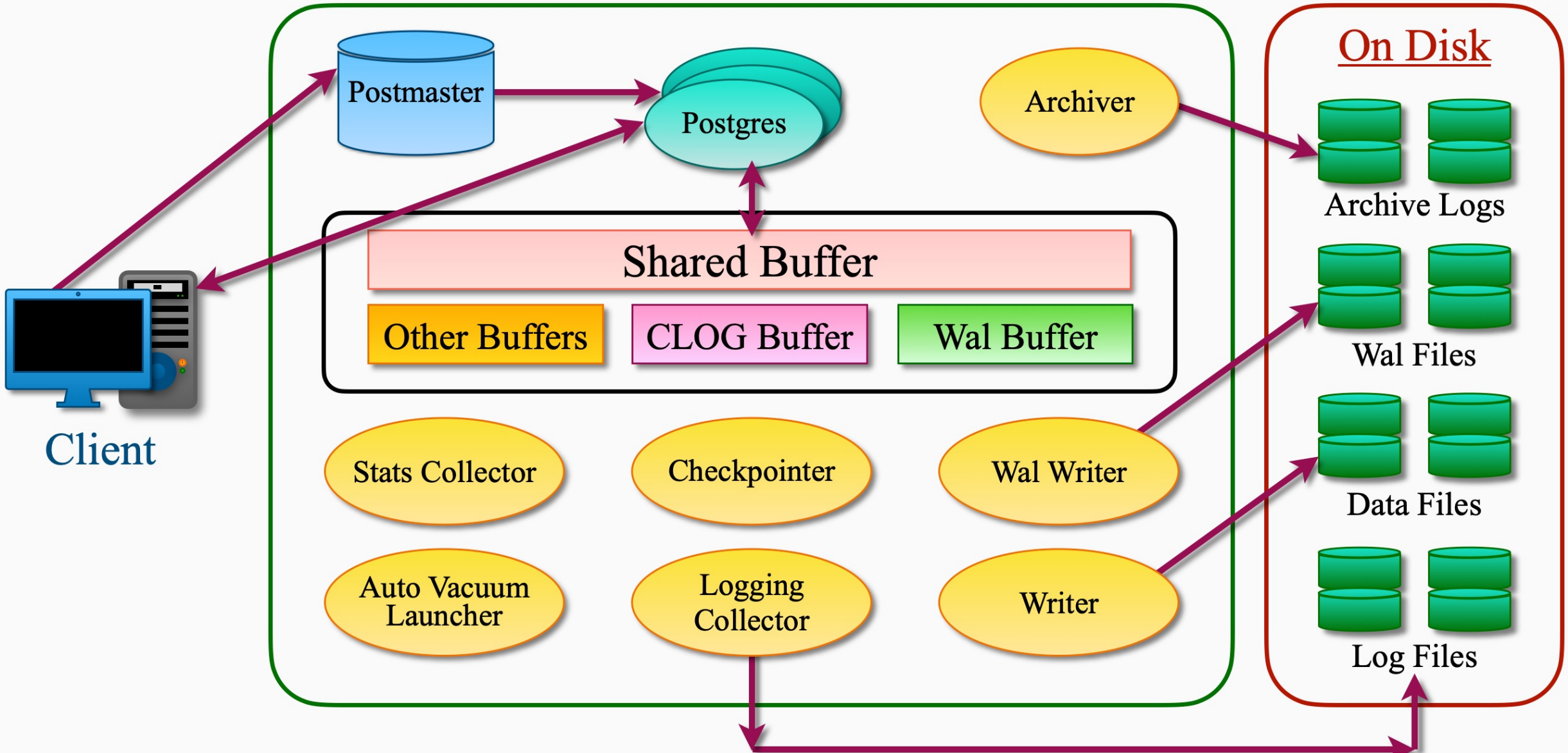
PostgreSQL vs Oracle: Key architectural differences

PostgreSQL installation & tools (pgAdmin, DBeaver)

Overview of pg_catalog, schemas, roles, and databases

PostgreSQL vs Oracle: Key architectural differences

PostgreSQL Process and Memory Architecture



PostgreSQL Architecture Key components

1. Postmaster Process (Supervisor)
2. Shared Memory Segments
3. Background Processes
4. Physical File Structure

1. Postmaster Process (Supervisor)

The **first process**

Started when PostgreSQL launches

1. Postmaster Process (Supervisor)

Acts as a **listener**, handles:

Authentication

Authorization

Spawning new backend processes

Monitoring and restarting failed processes

2. Shared Memory Segments

Memory areas used for

query execution,

data caching, and

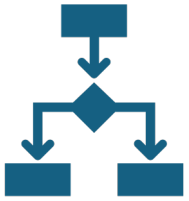
maintenance tasks.

Shared Buffers

Used for all **read/write operations**

Modified (dirty) pages are written to disk

by the **background writer**



Controlled by: shared_buffers

WAL Buffers (Write-Ahead Logging)

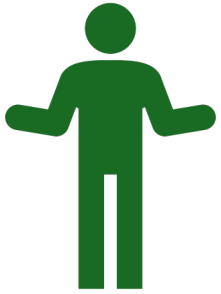
Temporarily store **metadata of changes**

Ensures **durability** and **crash recovery**

Controlled by: wal_buffers

CLOG Buffers (Commit Logs)

Track **transaction status**



(committed, aborted, etc.)

Shared across all sessions

Work Memory

Used for **sorting, hashing, joins**

Controlled by: work_mem

Separate allocation for each operation

Maintenance Work Memory

Used for **index creation,**

foreign key additions, etc.

Controlled by:

maintenance_work_mem

Temp Buffers

Used by **temporary tables**

Session-specific

3. Background Processes

Ensure

data consistency

help

maintain database health.

Background Writer

Writes

dirty pages

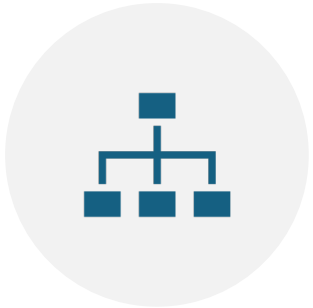
from shared buffers

to disk

Background Writer

CONTROLS:

BGWRITER_DELAY



BGWRITER_LRU_MAXPAGES

BGWRITER_LRU_MULTIPLIER

Checkpointer

Periodically **flushes**

dirty pages to disk

Essential for

crash recovery

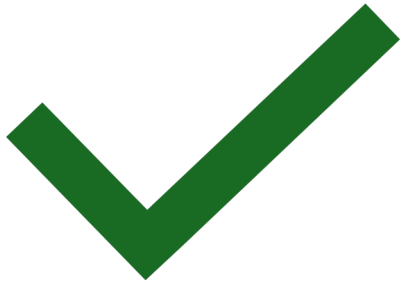
Checkpointer

Triggered by:

- Timeout (checkpoint_timeout)
- Manual CHECKPOINT command
- Backup processes
- max_wal_size limit

WAL Writer

Writes data from **WAL buffer to WAL files** on disk



Ensures write operations are **persistent before commit**

Autovacuum Launcher

Automates **VACUUM** and **ANALYZE**

Maintains **table statistics and bloat control**

Statistics Collector

Gathers usage metrics:

- Query counts
- Table scans
- Function usage

Statistics Collector

Controlled by:

- track_activities
- track_functions
- track_counts
- track_io_timing

Logging Collector

Captures server logs to files

Controlled by:

logging_collector = ON

log_directory = 'pg_log'

Archiver

Saves WAL segments to archive storage

for point-in-time recovery

Enabled during archive mode

4. Physical File Structure

PostgreSQL uses several file types

to store and manage data persistently:

Data Files

Store **actual user data**

Accessed via shared buffers

for reading/writing

WAL Files

Store **change logs** (Write-Ahead Logging)

Used for crash recovery

Log Files

Store **server logs** including

errors,

warnings, and

execution info

Archive Logs

Archived WAL segments

for **recovery purposes**

Workflow Summary

Client sends a query ➡️ received by Postmaster

Postmaster forks a backend (Postgres) process

Data retrieved from Shared Buffers

(or disk if not cached)

Workflow Summary

If **write operation**:

Change goes to Shared Buffer → WAL Buffer

WAL Writer commits to WAL file

Background Writer/Checkpointer writes to disk

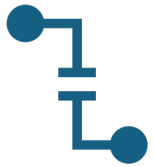
Statistics collected → Logging → Archiving if enabled

Conclusion

PostgreSQL's architecture is

robust, scalable, and

designed for high reliability,



fault-tolerance, and concurrency.

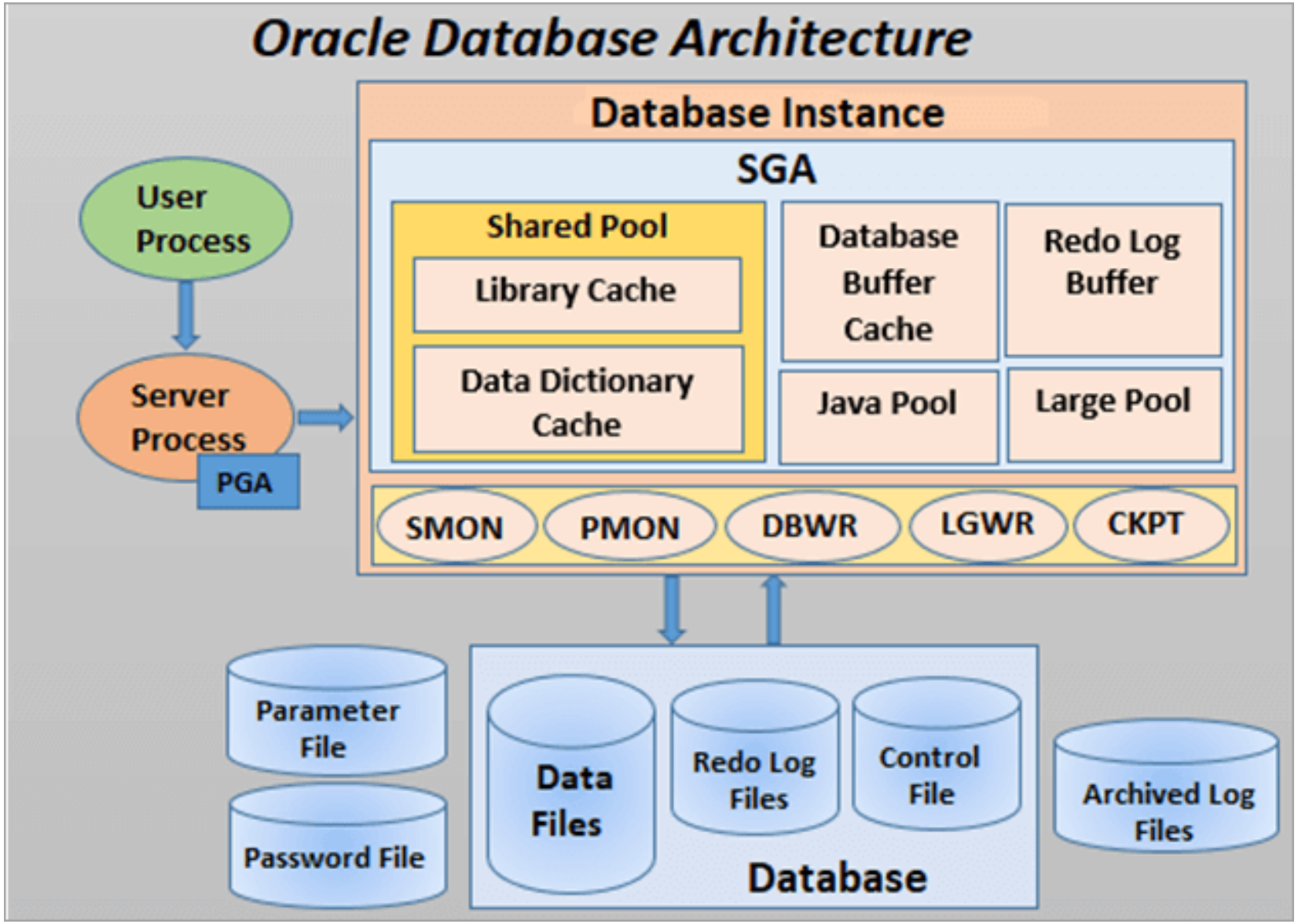
Conclusion

Its modular background processes and

memory components

ensure smooth operations and quick recovery,

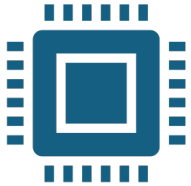
making it a strong choice for modern applications.


Oracle Database Architecture
Database Instance
SGA
User Process
Server Process
PGA
Shared Pool
Library Cache
Data Dictionary Cache
Database Buffer Cache
Redo Log Buffer
Java Pool
Large Pool
SMON
PMON
DBWR
LGWR
CKPT
Parameter File
Password File
Data Files
Redo Log Files
Control File
Database
Archived Log Files

Oracle Database Architecture

Oracle Database is a **multi-model**

relational database management system



known for its scalability, robustness, and

enterprise-grade features.

Oracle Database Architecture

Designed to manage

large amounts of data efficiently

using a combination of memory,

processes, files, and logical storage structures.

License & Philosophy

Feature	PostgreSQL	Oracle
License	Open Source (PostgreSQL License - similar to BSD)	Commercial (Proprietary License)
Philosophy	Extensibility, Standards Compliance, Community-Driven	Feature-Rich, Enterprise-Grade, Vendor Lock-in

Architecture Overview

Aspect	PostgreSQL	Oracle Database
Process Model	Multi-Process (One process per client connection + background processes)	Multi-Threaded (Single process with multiple threads + background processes)
Memory Architecture	Shared Buffers, Work Mem, WAL Buffers	SGA (System Global Area), PGA (Program Global Area), Redo Log Buffer

Architecture Overview

Aspect	PostgreSQL	Oracle Database
Storage Engine	Single storage engine	Multiple engines (Heap Organized Tables, Index Organized Tables, etc.)
Extensibility	Highly extensible (user-defined types, operators, functions, custom languages)	Limited extensibility (supports PL/SQL, but not new data types easily)

Transaction & Concurrency Control

Feature	PostgreSQL	Oracle
MVCC (Multi-Version Concurrency Control)	Native MVCC (Row-versioning, no read locks for SELECTs)	MVCC via Undo Segments (Undo data in rollback segments)
Locking	Row-level locks, advisory locks	Row-level locks, fine-grained locking, table-level partition locks
Isolation Levels	Read Committed, Repeatable Read, Serializable	Read Committed, Serializable, Read-Only Transactions

Replication & High Availability

Feature	PostgreSQL	Oracle
Replication	Streaming Replication, Logical Replication (built-in), 3rd party tools (e.g., Bucardo)	Oracle Data Guard (physical), Oracle GoldenGate (logical)
Sharding	External tools (Citus, pg_shard)	Native (Oracle Sharding)

Data Types & SQL Compliance

Feature	PostgreSQL	Oracle
Data Types	Rich: JSON/JSONB, Arrays, HSTORE, Range Types	Limited: JSON support added in 12c but less flexible
Standards	Very close to ANSI SQL compliance	SQL compliance + proprietary PL/SQL

Partitioning

Feature	PostgreSQL	Oracle
Partitioning	Declarative partitioning (since PostgreSQL 10)	Advanced partitioning (list, range, hash, composite)
Management	Manual in earlier versions; improved automation in recent versions	Fully automated, advanced features

Backup & Recovery

Feature	PostgreSQL	Oracle
Backup	pg_dump (logical), base backups + WAL archiving	RMAN (block-level backup), Data Pump (logical export/import)
Point-in-Time Recovery	WAL-based PITR	Redo/Undo Logs, Flashback Database

PL/SQL vs PL/pgSQL

Feature	PostgreSQL	Oracle
Procedural Language	PL/pgSQL (open), also supports Python, Perl, etc.	PL/SQL (proprietary, tightly integrated)
Triggers/Functions	Easily create functions in multiple languages	Primarily in PL/SQL

Performance & Optimization

Feature	PostgreSQL	Oracle
Optimizer	Cost-based, extensible, adaptive plans in progress	Highly advanced Cost-based Optimizer, adaptive plans, hints
Parallel Query	Supported but less mature	Advanced, with fine-grained control

Security & Auditing

Feature	PostgreSQL	Oracle
Authentication	SSL, GSSAPI, LDAP, Kerberos	Advanced (including fine-grained auditing, TDE, DB Vault)
Auditing	Basic logging and extensions (e.g., pgaudit)	Comprehensive built-in auditing

When to Use What?

PostgreSQL	Oracle
Startups, Open Source Projects	Large Enterprises, Legacy Systems
Cost-sensitive environments	High compliance, mission-critical
Customization & Extensions	Advanced features, support

Summary of Key Differences

PostgreSQL	Oracle
Open-source, community-driven	Proprietary, enterprise-focused
Process-based architecture	Thread-based shared memory architecture
Extensible with custom data types	Fixed feature set, but rich enterprise features
MVCC via row versioning	MVCC via undo segments
Free, with optional paid support	License-based, expensive but feature-rich

Final Thoughts

PostgreSQL: Innovation, Freedom, Extensibility 🚀

Oracle: Stability, Enterprise Support, Advanced Features 🏢

Choose based on your needs, not hype!

PostgreSQL vs Oracle vs SQL Server vs AWS RDS PostgreSQL

Area	PostgreSQL	Oracle	SQL Server	AWS RDS PostgreSQL
Licensing	Free	Very expensive	Expensive	Pay-as-you-use
MVCC	Heap-based	Undo-based	TempDB-based	Same as PostgreSQL
HA	Replication-based	RAC / DG	AlwaysOn	Managed Multi-AZ

PostgreSQL vs Oracle vs SQL Server vs AWS RDS PostgreSQL

Area	PostgreSQL	Oracle	SQL Server	AWS RDS PostgreSQL
OS Access	Full	Full	Full	❌ No
Vacuum / Cleanup	DBA-managed	Undo-managed	Auto	DBA-tuned
Cloud Fit	Excellent	Limited	Moderate	Excellent

Fidelity-Specific Insight

Oracle retained for

core settlement & legacy systems

PostgreSQL used for:

Investment reporting

Retirement platforms

Portfolio microservices

Fidelity-Specific Insight

RDS PostgreSQL used where:

Managed HA & backups are preferred

OS-level access is not required

Architecture Differences
Local PostgreSQL vs RDS PostgreSQL

Local PostgreSQL (Self-Managed)

DBA controls:	postgresql.conf	WAL behavior
Checkpoints	Vacuum tuning	OS, filesystem, storage

RDS PostgreSQL

AWS manages:

Instance lifecycle

Failover

Backups

Patching

RDS PostgreSQL

DBA still controls:

Schema design

Indexes

SQL

Vacuum tuning

Statistics

Query performance

Key DBA Mindset Shift

RDS removes sysadmin work

Not DBA responsibility

Fidelity Use Case

UK / India:

RDS used for regulated production workloads

China:

Managed environments reduce operational risk

Fidelity Use Case

DBAs still troubleshoot:

Slow queries

Blocking

Vacuum lag

Planner misestimates

MVCC – High-Level Concept

- Every UPDATE creates a **new row version**
- Old versions remain until cleaned by VACUUM
- Readers see a consistent snapshot
- Writers don’t block readers

Oracle vs PostgreSQL

Oracle	PostgreSQL
Undo segments	Heap-based versions
Automatic cleanup	Vacuum required
DBA tunes undo	DBA tunes vacuum

Fidelity Example

PORTFOLIO DASHBOARD (READ-HEAVY)

TRADE UPDATES (WRITE-HEAVY)

Fidelity Example

➡️ MVCC allows:

Real-time dashboards

Without blocking trade execution

⚠️ But poor vacuum

silent performance degradation

Important Differences DBAs Must Internalize

- No undo tablespaces → **VACUUM is mandatory**
- No RAC equivalent → replication-based HA
- No optimizer hints (by default)
- Planner accuracy depends heavily on statistics
- Autovacuum tuning is **non-negotiable**

Typical Oracle DBA Surprise

The database didn’t crash

but performance degraded slowly.

That’s usually **vacuum + statistics**,

not hardware.

Fidelity-Centric Summary

PostgreSQL is enterprise-ready

DBAs play a **more active performance role**

Same SQL knowledge applies, but **internals differ**

RDS simplifies ops, not tuning

PostgreSQL success = **MVCC + Vacuum + Planner mastery**

Fidelity Use Case

PostgreSQL used for **investment reporting,**

portfolio views, microservices

Oracle retained for **core settlement / legacy systems**

RDS PostgreSQL used for **managed production workloads**

Why PostgreSQL Fits These Workloads

PostgreSQL is ideal where workloads are:

Read-heavy

Highly concurrent

Evolving in schema

Cost-sensitive

Cloud-native or microservice-based

Typical Fidelity Workloads

- Portfolio dashboards
- Investment performance reports
- Fund analytics
- Customer-facing microservices
- Internal analytics platforms

DBA-Relevant Reasons

Requirement	Why PostgreSQL Works
High read concurrency	MVCC ensures readers don’t block writers
Frequent updates	No read locks on SELECT
Reporting joins	Strong optimizer + advanced indexing
Schema flexibility	JSONB + partial indexes
Cost control	No license cost
Global access	Works well with replication & read replicas

Tools Overview & Connectivity

psql vs pgAdmin (DBA usage patterns)

Connecting locally

Roles, databases, schemas

Pagila dataset overview

psql vs pgAdmin

psql

Command Line Interface (CLI)

pgAdmin 4

Web-based GUI administration tool

psql (CLI) — DBA Perspective

Lightweight, scriptable,

terminal-based client

Similar to sqlplus (Oracle) or

sqlcmd (SQL Server)

Why DBAs Prefer psql

Fast during incidents

Scriptable for automation

Works over SSH / jump hosts

Minimal overhead

Common DBA Use Cases at Fidelity

- Incident troubleshooting (locks, slow queries)
- Quick validation during deployments
- Running scripts across environments
- Connecting to RDS PostgreSQL from bastion hosts

Example

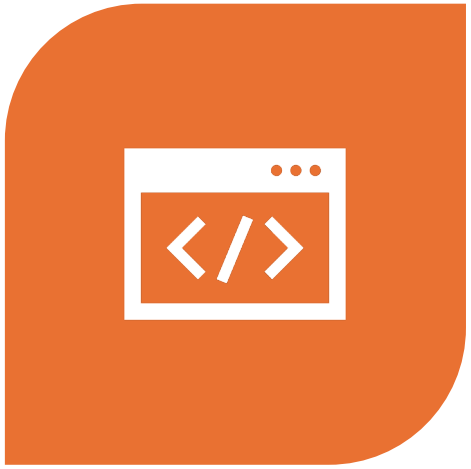
```
psql -h localhost -U postgres
```

```
\l
```

```
\dt
```

```
SELECT * FROM pg_stat_activity;
```

pgAdmin 4 (GUI) — DBA Perspective



WEB-BASED GRAPHICAL TOOL

SIMILAR TO ORACLE SQL DEVELOPER / SSMS

Why DBAs Use pgAdmin

Visual query plans

Index and schema browsing

Explaining query behavior to teams

Ad-hoc analysis and learning

Common DBA Use Cases at Fidelity

Query plan analysis for reporting teams

Reviewing index usage

Supporting junior DBAs / developers

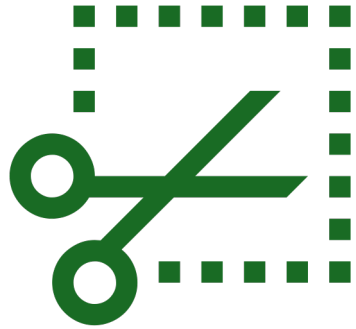
Browsing objects in unfamiliar schemas

psql vs pgAdmin — Clear Comparison

Area	psql	pgAdmin
Speed	⭐⭐⭐⭐⭐	⭐⭐⭐
Automation	Excellent	Limited
Learning Curve	Steep	Easy
Incident Handling	Best choice	Not ideal
Visual Plans	❌	✅
Production Usage	Very common	Selective

DBA Rule of Thumb

psql for operations.



pgAdmin for analysis and explanation.

Connecting Locally to PostgreSQL

Local Connection Model

PostgreSQL supports:

TCP/IP connections

Local socket connections

Connecting Locally to PostgreSQL

- **Typical Local Connection**

- psql -h localhost -p 5432 -U postgres

What Happens Internally

Connection request hits **postmaster**

A backend process is forked

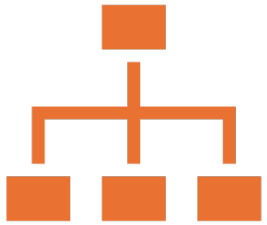
Session is authenticated

Backend process handles all queries

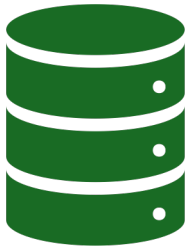
Fidelity Context

Same connection logic applies for:	Local PostgreSQL	AWS RDS PostgreSQL
Only difference in RDS:	SSL enforced	No OS access

Roles, Databases, and Schemas



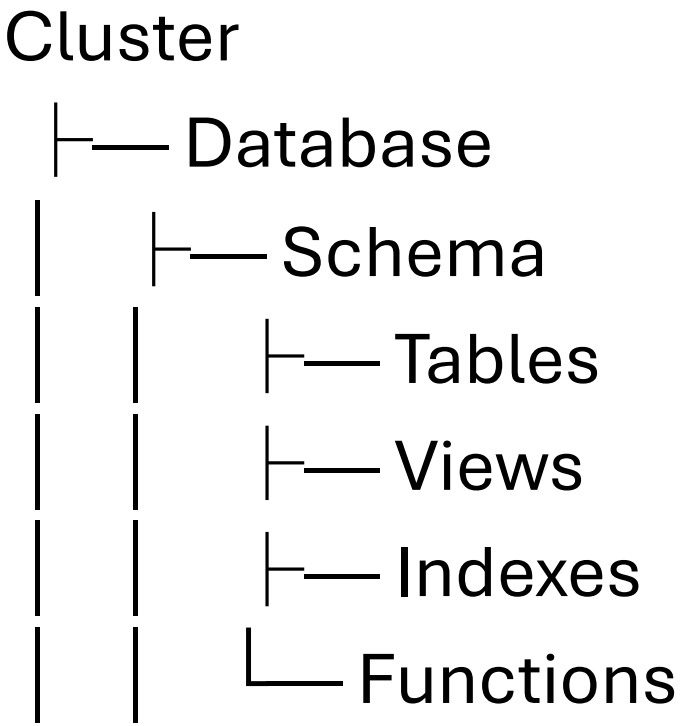
This hierarchy is **critical for**



Oracle / SQL Server DBAs

to understand correctly.

PostgreSQL Object Hierarchy



Roles (Users & Groups)

Roles for:

Authentication

Authorization

A role can:

Login (user)

Own objects

Be granted to other roles (group)

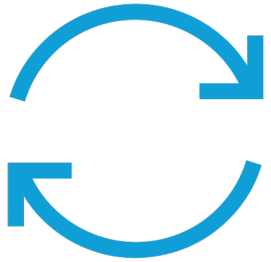
Oracle Comparison

Oracle	PostgreSQL
User	Role with LOGIN
Role	Role (same concept)

Databases

Logical containers

One connection → one database



Cannot join across databases (without FDW)

DBA Impact

Multiple databases share:

Memory

WAL

Background processes

Schemas

What Schemas Do

Logical namespaces inside a database

Organize objects

Control access

Oracle Comparison

Oracle	PostgreSQL
Schema = User	Schema ≠ User

Pagila Dataset Overview

What Pagila Is

Standard PostgreSQL sample database

Equivalent to:

Oracle HR schema

SQL Server AdventureWorks

What Pagila Contains

Customers

Transactions

Inventory

Time-based data

Multiple joins and indexes

Why Pagila Is Used in This Training

Demonstrates:

- Joins
- Index usage
- Query plans
- MVCC behavior
- Vacuum effects

Fidelity Mapping

Pagila tables simulate:

Customers → Investors

Rentals → Transactions

Payments → Cash flows

Inventory → Investment products

Summary for Fidelity DBA Audience

PostgreSQL tools are **simple but powerful**

psql is the **primary DBA tool**

pgAdmin is a **supporting analysis tool**

Roles ≠ schemas (common Oracle DBA mistake)

Pagila provides realistic, safe practice data

Hands-On Lab

- Connect using psql and pgAdmin
- Create roles, schemas, objects
- Load Pagila schema + data
- Run basic queries

Hands-On Lab

- Measure performance using:
- \timing
- EXPLAIN / EXPLAIN ANALYZE
- Explore system catalogs:
- pg_class
- pg_stat_activity

Connect to PostgreSQL

Database: postgres

Username: postgres

Password: your password

Test Connection & Finish

Summary

Tool	Purpose
PostgreSQL 16	Database Server
pgAdmin 4	Web-based GUI for PostgreSQL
DBeaver	Universal SQL Client & ER Diagram Tool

Useful Links

PostgreSQL Docs: https://www.postgresql.org/docs/16/

pgAdmin Docs: https://www.pgadmin.org/docs/

DBeaver Docs: https://dbeaver.io/docs/

Overview of pg_catalog, schemas, roles, and databases

Surendra Panpaliya

Databases in PostgreSQL

Self-contained unit of data.

Each database has its own

schemas, tables,

users (roles), and configurations

Key Points

PostgreSQL is a multi-database system.

Each database is isolated from others

(no cross-database queries directly).

Default databases

postgres → Default working database

template1 → Template for new databases

template0 → Clean template (minimal)

Schemas

Namespace inside a database.

Think of it as a folder to organize

tables, views, functions, etc.

Key Points

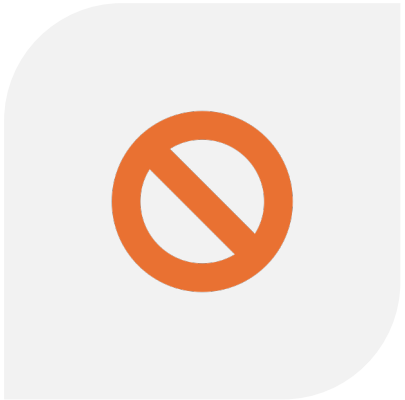
Each database can have **multiple schemas**.

Default schema: public

Objects are referenced as:

schema_name.object_name

Why Schemas?



AVOID NAME CONFLICTS

LOGICAL SEPARATION OF DATA

SUPPORT FOR MULTI-TENANT APPLICATIONS

Command Examples

```
-- Create a schema
CREATE SCHEMA sales;

-- Create table inside schema
CREATE TABLE sales.orders (id SERIAL PRIMARY KEY, item TEXT);

-- Access table
SELECT * FROM sales.orders;
```

pg_catalog

A system schema

Stores PostgreSQL

internal metadata.

Pg_catalog

pg_catalog is a built-in system schema in PostgreSQL

contains all the system tables,

views, data types, functions, and

operators used internally by PostgreSQL

to manage the database.

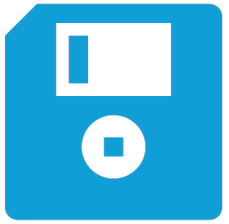
Contains

Object	Purpose
pg_class	Info about tables, indexes, sequences
pg_user	View of roles with login
pg_roles	Info about all roles
pg_namespace	Info about schemas
pg_tables	List of tables
pg_indexes	List of indexes

Use Case:

Query database metadata

Understand internal PostgreSQL structures



System operations, backups, monitoring

Example Queries:

```
-- List all tables using pg_catalog
SELECT tablename FROM pg_catalog.pg_tables WHERE schemaname = 'public';


-- List all roles
SELECT rolname FROM pg_catalog.pg_roles;
```

Roles

A **Role** in PostgreSQL = **User Account + Group**

Roles manage **authentication & authorization**.

Types of Roles:

Role	Description
Login Role	Can log in (CREATE ROLE username LOGIN;)
Group Role	Used for permissions, no login

Privileges Controlled By Roles:

CONNECT	CONNECT (to database)
CREATE	CREATE (schemas, tables)
SELECT	SELECT, INSERT, UPDATE, DELETE (on tables)
EXECUTE	EXECUTE (functions)

Role Management Examples

-- Create role with login

CREATE ROLE devuser LOGIN PASSWORD 'mypassword';

-- Grant role permissions

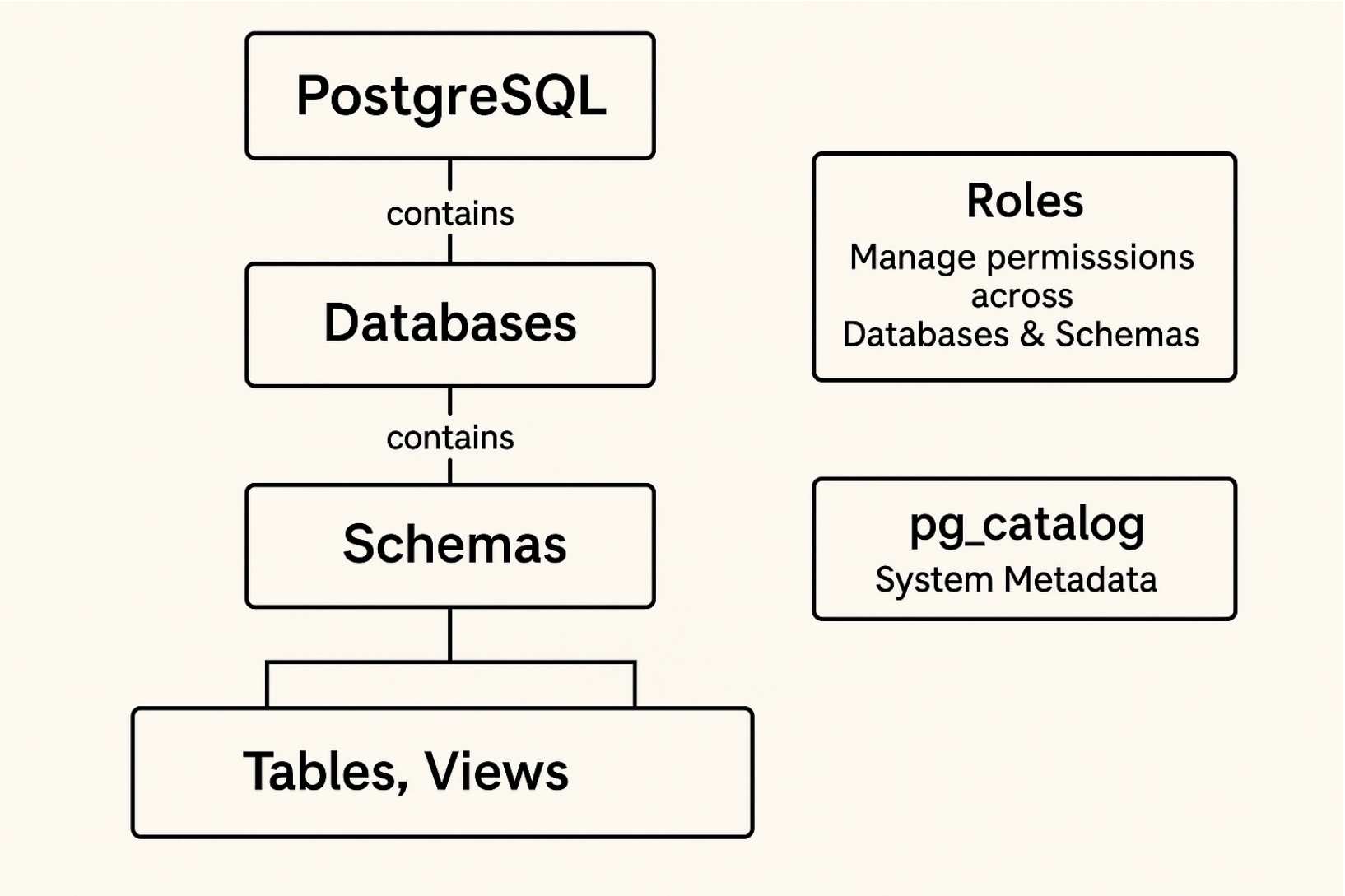
GRANT CREATE ON DATABASE mydb TO devuser;

Role Management Examples

```
-- Create group role
CREATE ROLE analyst;

-- Assign user to group
GRANT analyst TO devuser;
```

Relationship Diagram



Summary Table

Concept	Purpose
Database	Top-level data container
Schema	Namespace within a database
pg_catalog	Stores metadata about objects
Role	User and permission management

Real-World Analogy

PostgreSQL Concept	Analogy
Database	Apartment Building
Schema	Individual Apartment
Table/View	Furniture inside an apartment
Role	Security Guard controlling access
pg_catalog	Building Management Records

Surendra Panpaliya

Founder and CEO

GKTCS Innovations

https://www.gktcs.com